Draft NISTIR 8063

Primitives and Elements of Internet of Things (IoT) Trustworthiness

Jeffrey Voas

National Institute of
Standards and Technology
U.S. Department of Commerce

Draft NISTIR 8063

Primitives and Elements of Internet of Things (IoT) Trustworthiness

Jeffrey Voas
Computer Security Division
Information Technology Laboratory

February 2016

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

National Institute of Standards and Technology Internal Report 8063
21 pages (February 2016)

Public comment period: *February 16, 2016* through *March 17, 2016*

All comments are subject to release under the Freedom of Information Act (FOIA).

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: iot@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in federal information systems.

Abstract

System primitives allow formalisms, reasoning, simulations, and reliability and security risk-tradeoffs to be formulated and argued. In this work, five core primitives belonging to most distributed systems are presented. These primitives apply well to systems with large amounts of data, scalability concerns, heterogeneity concerns, temporal concerns, and elements of unknown pedigree with possible nefarious intent. These primitives form the basic building blocks for a Network of 'Things' (NoT), including the Internet of Things (IoT). This report offers an underlying and foundational science to IoT. To our knowledge, the ideas and the manner in which IoT is presented here is unique.

Keywords

big data; composability; distributed system; Internet of Things (IoT); Network of Things (NoT); reliability; security; trust; trustworthiness.

Note to Readers: This report describes research on creating a vocabulary, namely primitives and elements, for composing IOT. We present five primitives and six elements that form a design catalogue that can support trustworthiness. We envision their application to use cases, ontologies, formalisms, and other methods to specific IOT projects. We see this as early research and earnestly seek feedback on the merits, utility, and feasibility of such a vocabulary.

Table of Contents

List of Figures

List of Tables

1 Introduction

From agriculture, to manufacturing, to smart homes, and to healthcare, there is value in having numerous sensory devices connected to larger infrastructures.

However the current Internet of Things (IoT) landscape presents itself as a mix of jargon, consumer products, and unrealistic predictions. There is no formal, analytic, or even descriptive set of the building blocks that govern the operation, trustworthiness, and lifecycle of IoT. This vacuum between the hype and the science, if a science exists, is evident. Therefore, a composability model and vocabulary that defines principles common to most, if not all networks of things, is needed to address the question: “what is the science, if any, underlying IoT?”

For clarification, this paper uses two acronyms, IoT and NoT (Network of Things), interchangeably —the relationship between NoT and IoT is subtle. IoT is an instantiation of a NoT, more specifically, IoT has it’s ‘things’ tethered to the Internet. A different type of NoT could be a Local Area Network (LAN), with none of its’ ‘things’ connected to the Internet. Social media networks, sensor networks, and the Industrial Internet are all variants of NoTs. This differentiation in terminology provides ease in separating out use cases from varying vertical and quality domains (e.g., transportation, medical, financial, agricultural, safety-critical, security-critical, performance-critical, high assurance, to name a few). That is useful since there is no *one*, static IoT.

Primitives are building blocks that offer the possibility of an answer to the aforementioned question by allowing comparisons between NoTs. We use the term primitive to represent smaller pieces from which larger blocks or systems can be built. For example, in software coding, primitives typically include the arithmetic and logical operations (plus, minus, and, or, etc.). It is outside the scope of this writing to address issues such as what is small, smaller, smallest, atomic, etc.

Primitives offer a unifying vocabulary that allows for composition and information exchange among differently purposed networks. They offer clarity regarding more subtle concerns, including interoperability, composability, and continuously-binding assets that come and go on-the-fly. Because no simple, actionable, and universally-accepted definition for IoT exists, the model and vocabulary proposed here reveals underlying foundations of the IoT, i.e., they expose the ingredients that can express how the IoT *behaves*, without defining IoT. This offers insights into issues specific to trust.

Further, we employ a paraphrased, general definition for a *distributed system*: a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.[1] NoTs satisfy this definition. Thus we consider IoT to be one type of a NoT and a NoT to be one type of a distributed system.

[1] George Coulouris et al., *Distributed Systems: Concepts and Design*, 5th ed. (Boston: Addison-Wesley, 2011), 2.

2 The Primitives

The *primitives* we propose are: 1) **Sensor**, 2) **Aggregator**, 3) **Communication channel**, 4) ***e*Utility**, and 5) **Decision trigger**. Each primitive, along with its definition, assumptions, properties, and role, is presented. We employ a data-flow model, captured as a sequence of four figures, to illustrate how primitives, when composed in a certain manner, could impact a confidence in trustworthiness. Although this model may seem overly abstract at first glance, its simplicity offers a certain elegance by not over complicating IoT's handful of building blocks.

2.1 Primitive #1: Sensor

A *sensor* is an electronic utility that digitally measures physical properties such as temperature, acceleration, weight, sound, etc. Basic properties, assumptions, and general statements about sensor include:

1. Sensors are physical.
2. Sensor output is data; in our writings, $s_1 \rightarrow d_1$ means that sensor 1 has produced a piece of data that is numbered 1. Likewise, $s_2 \rightarrow d_2$ means that sensor 2 has produced a piece of data that is numbered 2.
3. Sensors may have little or no software functionality and computing power; more advanced sensors may have software functionality and computing power.
4. Sensors will likely be heterogeneous, from different manufacturers, and collect data, with varying levels of data integrity.
5. Sensors will have operating geographic locations that may change.
6. Sensors may provide surveillance. Cameras and microphones are sensors.
7. Sensors may have an owner(s) who will have control of the data their sensors collect, who is allowed to access it, and when.
8. Sensors will have pedigree – geographic locations of origin and manufacturers. Pedigree may be unknown and suspicious.
9. Sensors may fail continuously or fail intermittently.
10. Sensors may be cheap, disposable, and susceptible to wear-out over time; here, building security into a specific sensor will rarely be cost effective. However there will differentials in security, safety, and reliability between consumer grade, military grade, industrial grade, etc.
11. Sensors may return no data, totally flawed data, partially flawed data, or correct/acceptable data.

12. Sensors are expected to return data in certain ranges, e.g., [1 … 100]. When ranges are violated, rules may be needed on whether to turn control over to a human or machine when ignoring out-of-bounds data is inappropriate.

13. Sensor repair is likely handled by replacement.

14. Sensors may be acquired off-the-shelf.

15. Sensors release data that is event-driven, driven by manual input, or released at pre-defined times.

16. Sensors may have a level of data integrity ascribed (Section 2.2.2).

17. Sensors may have their data encrypted to void some security concerns.

18. Sensor data may be leased to multiple NoTs. A sensor may have multiple recipients of its data.

19. The frequency with which sensors release data impacts the data's currency and relevance. Sensors may return valid data at an incorrect rate/speed.

20. Sensor data may be 'at rest' for long periods of time; sensor data may become *stale.*

21. A sensor's resolution may determine how much information is provided.

22. Security is a concern for sensors if they or their data is tampered with or stolen.

23. Reliability is a concern for sensors.

2.2 Primitive #2: Aggregator

An *aggregator* is a software implementation based on mathematical function(s) that transforms groups of raw data into *intermediate* data. Basic properties, assumptions, and general statements about aggregator include:

1. Aggregators are likely virtual due the benefit of changing implementations quickly and increased malleability. A situation may exist where aggregators are physically manufactured, e.g., a FPGA or hard-coded aggregator that is not programmable, similar to an *n*-version voter.

2. Aggregators are assumed to lack computing horsepower, however this assumption can be relaxed by changing the definition and assumption of virtual to physical, e.g. firmware, microcontroller or microprocessor. Aggregators will likely use weights (Section 2.2.2) to compute intermediate data.

3. Aggregators have two actors that make them ideal for consolidating large volumes of data into lesser amounts: Clusters (Section 2.2.1), and Weights (Section 2.2.2). Aggregator is the *big data processor* within IoT.
4. Intermediate data may suffer from some level of *information loss*.
5. For each cluster (Section 2.2.1) there should be an aggregator or set of potential aggregators.
6. Aggregators are executed at a specific time and for a fixed time interval.
7. Aggregators may be acquired off-the-shelf.
8. Security is a concern for aggregators (malware or general defects) and for the sensitivity of their aggregated data.
9. Reliability is a concern for aggregators (general defects).

2.2.1 Actor #1: Cluster

A *cluster* is an abstract grouping of sensors that can appear and disappear instantaneously. Basic properties, assumptions, and general statements about cluster include:

1. Clusters are abstractions of a set of sensors along with the data they output—clusters may be created in an *ad hoc* manner or organized according to fixed rules.
2. Clusters are not inherently physical.
3. C_i is essentially a *cluster* of the sensor data from $n \geq 1$ sensors, $\{d_1, d_2, d_3, \ldots, d_n\}$.
4. C_i may share one or more sensors with C_k, where $i \neq k$.
5. *Continuous-binding* of a sensor to a cluster may result in little ability to mitigate trustworthiness concerns if the binding is *late*.
6. Clusters are malleable and can change their collection of sensors and their data.
7. How clusters are composed is dependent on what mechanism is employed to aggregate the data, which ultimately impacts the purpose and requirements of a specific NoT.

Note assumptions 4 and 6 above; these two assumptions are subtly important – they relate to business competition.

2.2.2 Actor #2: Weight

Weight is the degree to which a particular sensor's data will impact an aggregator's computation. Basic properties, assumptions, and general statements about weight include:

1. A weight may be hardwired or modified on the fly.
2. A weight may be based on a sensor's perceived trustworthiness, e.g., based on who is the sensor's owner, manufacturer, geographic location of manufacture, geographic location where the sensor is operating, sensor age or version, previous failures or partial failures of sensor, sensor tampering, sensor delays in returning data, etc. A weight may also be based on the value of the data, uniqueness, relation to mission goals, etc.
3. Different NoTs may leverage the same sensor data and re-calibrate the weights per the purpose of a specific NoT.
4. Aggregators may employ artificial intelligence to modify their clusters and weights.
5. Weights will affect the degree of information loss during the creation of intermediate data.
6. Security is probably not a concern for weights unless they are tampered with.
7. The appropriateness (or correctness) of the weights is crucial for the purpose of a NoT.

A simple aggregator might implement the summation

$$\sum_{i=1}^{x} d_i$$

divided by *x*, where the weight for each data point is *uniform*.

2.3 Primitive #3: Communication Channel

A *communication channel* is a medium by which data is transmitted (e.g., physical via USB, wireless, wired, verbal, etc.). Basic properties, assumptions, and general statements about communication channel include:

1. Communication channels move data between computing and sensing.
2. Since data is the "blood" of a NoT, communication channels are the "veins" and "arteries".
3. Communication channels will have a physical or virtual aspect to them, or both. For example protocols and associated implementations provide a virtual dimension, cables provide a physical dimension.
4. Communication channel dataflow may be unidirectional or bi-directional. There are a number of conditions where an aggregator might query more advanced sensors, or potentially recalibrate them in some way (e.g., request more observations per time interval).

5. No standardized communication channel protocol is assumed; a specific NoT may have multiple communication protocols between different entities.

6. Communication channels may be wireless.

7. Communication channels may be an offering (*service* or *product*) from third-party vendors.

8. Communication channel *trustworthiness* may make sensors appear to be failing when actually the communication channel is failing.

9. Communication channels are prone to disturbances and interruptions.

10. *Redundancy* can improve communication channel reliability.

11. Performance and availability of communication channels will greatly impact any NoT that has time-to-decision requirements (see the Decision trigger primitive in Section 2.5).

12. Security and reliability are concerns for communication channels.

In Figure 1, 15 sensors are shown – the blue sensors indicate that 2 sensors are ‘somehow’ failing at specific times, that is, they are not satisfying their purpose and expectations. As mentioned earlier, there could be a variety of sensor failure modes, some temporal, and some related to data quality. Further the temporal failure modes for sensors may be actually a result of the transport of that data failing, and not the sensors. Consider also that the two failing sensors in Figure 1 should probably be assigned lower weights. Figure 1 also shows the 15 sensors clustered into 3 clusters with 5 unique sensors assigned to each. Figure 1 shows the data coming out from each of the three clusters as being inputted to 3 corresponding aggregators. It is now the responsibility of the 3 aggregators to turn those 15 sensor inputs into 3 intermediate data points.

Note the close relationship between clusters and aggregators. For example, in Figure 1, aggregator C_1 might be determining how busy restaurant *A* is. Five independent sensors in *A* could be taking pictures from inside and outside (parking lot) of *A*, room temperature measurement in the kitchen, motion detectors from the dining area, sound and volume sensors, light detectors, etc. So while the sensors are certainly not homogeneous, their data is processed to make a new piece of data to address one question with possible results such as is the restaurant busy, not busy, closed, etc. And aggregators C_2 and C_3 might be doing the same for restaurants *B* and *C* respectively.

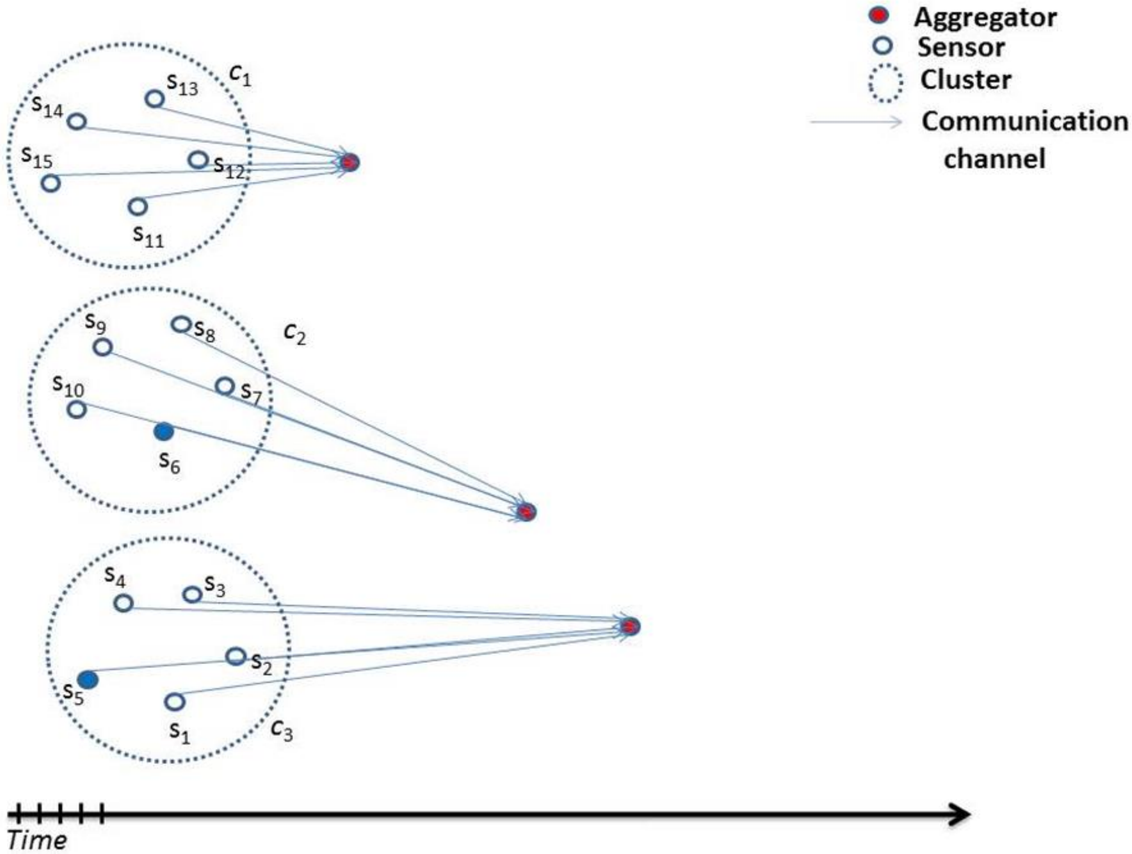

Figure 1: The first three primitives

2.4 Primitive #4: *e*Utility (external utility)

An *e*Utility (external utility) is a software or hardware product or service. Basic properties, assumptions, and general statements about *e*Utility include:

1. *e*Utilities execute processes or feed data into the overall dataflow of a NoT.
2. *e*Utilities may be acquired off-the-shelf.
3. *e*Utilities may include databases, mobile devices, misc. software or hardware systems, clouds, computers, CPUs, etc. The *e*Utility primitive can be subdivided, and probably should be decomposed as this model becomes less abstract.
4. *e*Utilities, such as clouds, provide computing power that aggregators may not have.
5. A human may be viewed as a *e*Utility.
6. Data supplied by a *e*Utility may be weighted.

7. An *e*Utility may be counterfeit; this is mentioned later in element Device_ID (Section 3).

8. Non-human *e*Utilities may have Device_IDs; Device_IDs may be crucial for authentication.

9. Security and reliability are concerns for *e*Utilities.

Figure 2 illustrates the use of two cloud *e*Utilities executing the functions of five aggregator implementations. (Different clouds could be from different cloud vendors.) Figure 2 shows the addition of one non-cloud *e*Utility, eU_1 (a laptop).

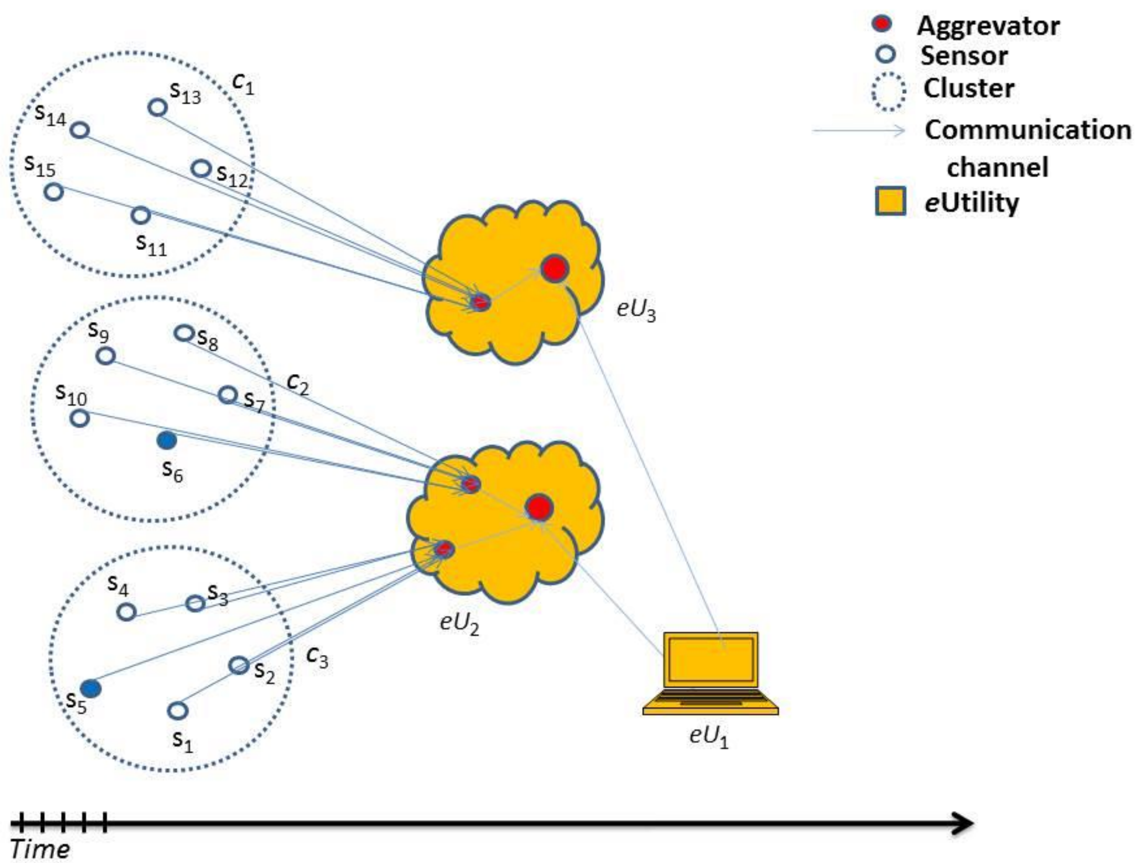

Figure 2: *e*Utility

2.5 Primitive #5: Decision Trigger

A *decision* trigger creates the final result(s) needed to satisfy the purpose, specification, and requirements of a specific NoT. Basic properties, assumptions, and general statements about decision trigger include:

1. A decision trigger is a pre-condition that must be TRUE before a NoT takes action. As shown in Figure 3, D = $f(\mathbf{x}, \mathbf{y})$, determines whether a particular action is taken. Put simply, D = $f(\mathbf{x}, \mathbf{y})$ abstractly defines the end-purpose of a NoT.

2. A decision trigger should have a corresponding virtual implementation.

3. A decision trigger may have a unique owner.

4. Decision triggers may be acquired off-the-shelf or homegrown.

5. Decision triggers are executed at specific times and may occur continuously as new data becomes available.

6. Decision trigger results may be predictions.

7. Decision trigger results may control actuators[2] or other transactions (see Figure 3 and Figure 4).

8. If a decision trigger feeds data signals into an actuator, then the actuator may be considered as a *e*Utility if the actuator feeds data back into the NoT.

9. A decision trigger may feed its output back into the NoT creating a feedback loop (See Figure 4).

10. It is fair to view a decision trigger as an **if-then** rule, although they will not all have this form.

11. The workflow up to decision trigger execution may be partially parallelizable.

12. Failure to execute decision triggers at time t_x may occur due to tardy data collection, inhibited sensors or *e*Utilities, inhibited communication channels, low performance aggregators, and a variety of other subsystem failure modes.

13. Economics and costs play a role in the quality of the decision trigger's output.

14. There may be intermediate decision triggers at any point in a NoT's workflow.

15. Decision triggers act similarly to aggregators and can be viewed as a special case of aggregator.

16. Security is a concern for decision triggers (malware or general defects).

[2] "A device for moving or controlling a mechanism or system. It is operated by a source of energy, typically electric current, hydraulic fluid pressure, or pneumatic pressure, and converts that energy into motion. An actuator is the mechanism by which a control system acts upon an environment. The control system can be simple (a fixed mechanical or electronic system), software-based (e.g. a printer driver, robot control system), or a human or other agent." [Stouffer 2015, p. B-1)]

17. Reliability is a concern for decision triggers (general defects). Decision triggers could be inconsistent, self-contradictory, and incomplete. Understanding how bad data propagates to affects decision triggers is paramount. Failure to execute decision triggers at time t_x may have undesired consequences.

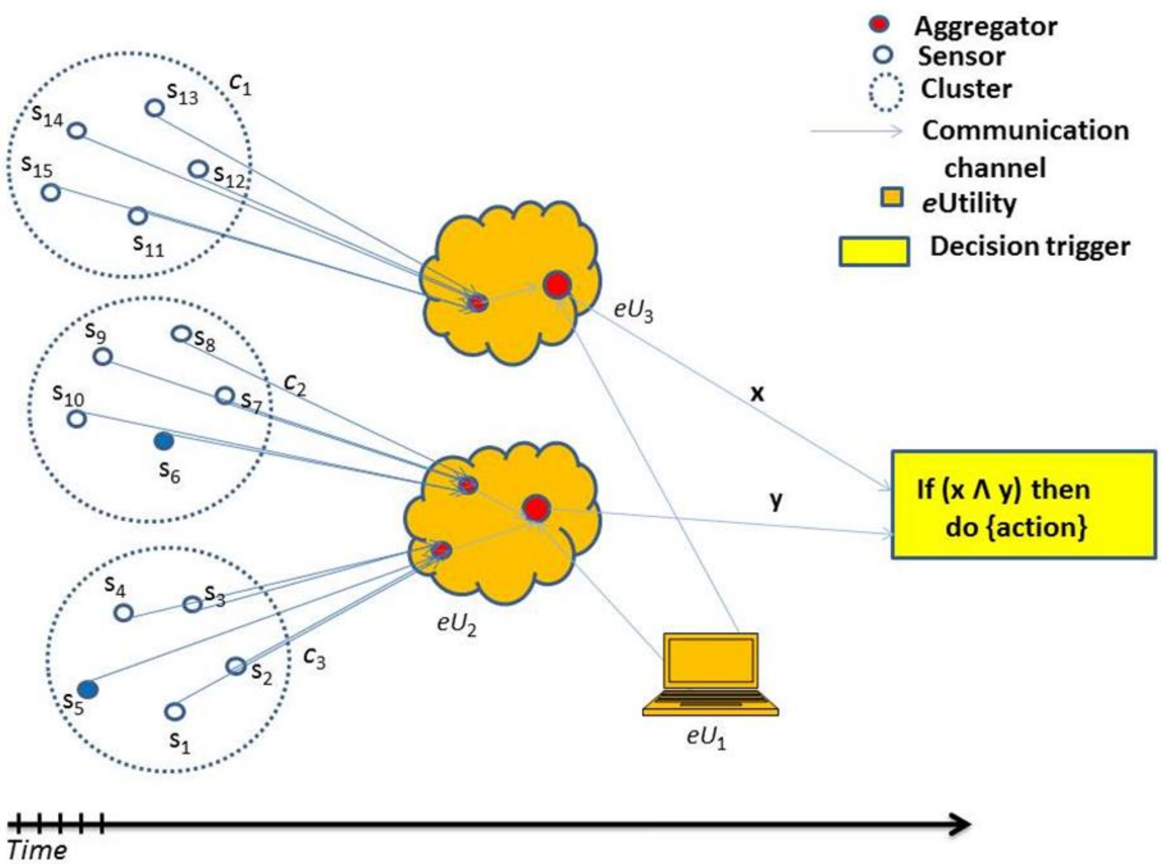

Figure 3: Decision trigger

Going back to our restaurant example, if C_2 did something similar for restaurant *B* and C_3 for restaurant *C*, and the laptop sent in data concerning the calendar and times when *A*, *B*, and *C* were open, then variables **x** and **y** in Figure 3 might be a data point as to whether these restaurants had customers during their open-for-business times. And obviously **x** and **y** could be refreshed as often as desired. The output of the decision trigger might be valuable information for a competing restaurant or a corporation if *A*, *B*, and *C* were parts of a restaurant brand.

Figure 4 shows an alternative to any suggestion that this model of a NoT's dataflow is necessarily uni-directional; it depicts a decision trigger that actually feeds its results back into the NoT, creating a continuous feedback loop. So for example if new sensor data were fed continuously into a NoT's workflow, that data can be combined with the results of previous decision trigger outputs to create updated decision trigger results at later points in time.

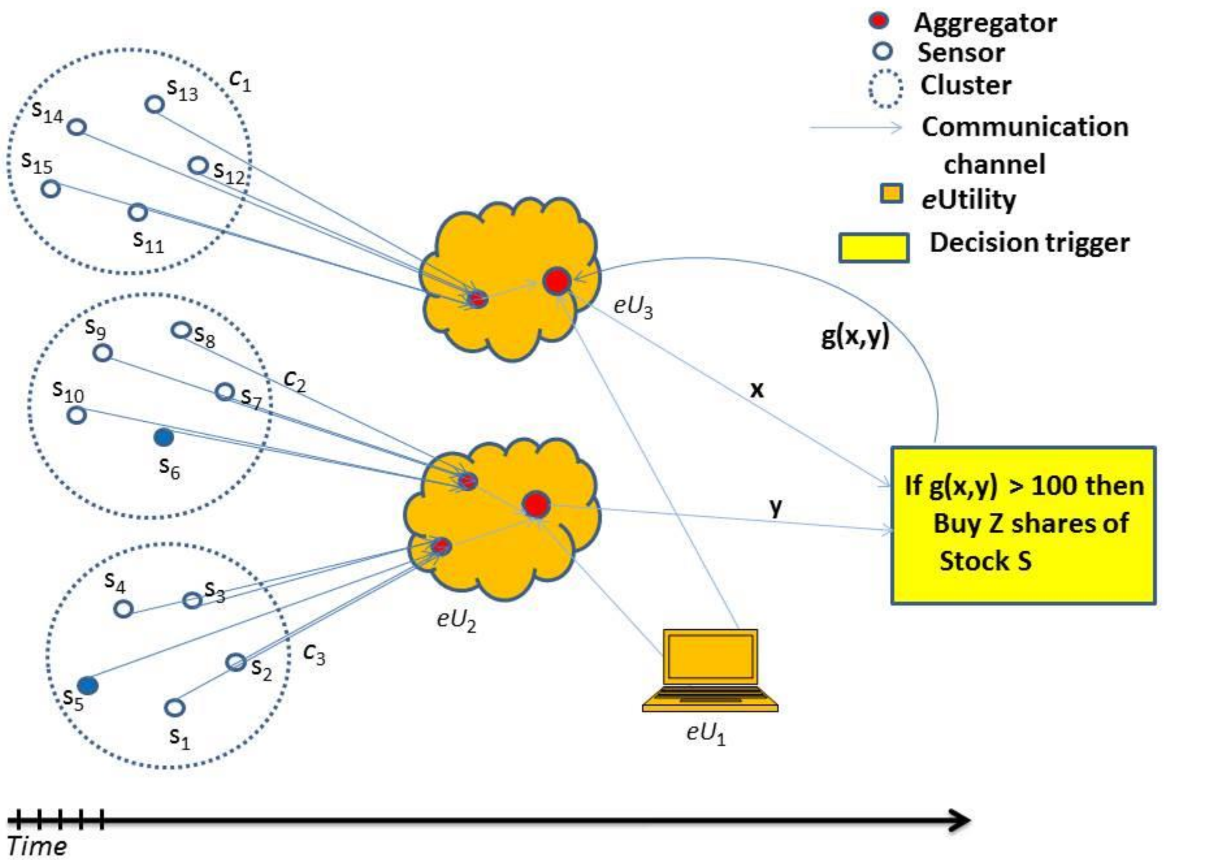

Figure 4: Decision trigger with feedback

2.6 Additional Notes on the Primitives

Now, a few additional points concerning the interplay and relationship between the five are as follows. First, sensor feeds aggregator. Secondly, aggregator executes on elements in *e*Utility. Thirdly, communication channel are the veins that connect sensor, aggregator, *e*Utility, and decision trigger with data between them. And fourth, sensor, aggregator, communication channel, *e*Utility, and decision trigger all have events firing at specific times; a large challenge for IoT and NoTs is to keep events in sync.

3 The Elements

To complete our model, we propose six elements: *environment*, *cost*, *geographic location*, *owner*, *Device_ID,* and *snapshot* that although are not primitives, are key players in trusting NoTs. These elements play a major role in fostering the degree of trustworthiness[3] that a specific NoT can provide.

1. **Environment** – The universe that all primitives in a specific NoT operate in; this is essentially the *operational profile* of a NoT. An analogy is the various weather profiles that an aircraft operates in or a particular factory setting that a NoT operates in. This will likely be very difficult to correctly define.

2. **Cost** – The expenses, in terms of time and money, that a specific NoT incurs in terms of the non-mitigated reliability and security risks; additionally, the costs associated with each of the primitive components needed to build a NoT. Cost is an estimation or prediction. Cost drives the design decisions in building a NoT.

3. **Geographic location –** Physical place where a sensor or *e*Utility operates or was manufactured. Manufacturing location is a supply chain trust issue. Note that the operating location may change over time. Note that a sensor's or *e*Utility's geographic location along with communication channel reliability may affect the dataflow throughout the workflow in a timely manner. Geographic location determination may sometimes not be possible.

4. **Owner** - Person or Organization that owns a particular sensor, communication channel, aggregator, decision trigger, or *e*Utility. There can be multiple owners for any of these five. Note that owners may have nefarious intentions that affect overall trust. Note further that owners may remain anonymous.

5. **Device_ID** – A unique identifier for a particular sensor, communication channel, aggregator, decision trigger, or *e*Utility. This will typically originate from the originator of the entity, but it could be modified or forged.

6. **Snapshot** – an instant in time. Basic properties, assumptions, and general statements about snapshot include:

    a. Because a NoT is a distributed system, different events, data transfers, and computations occur at different snapshots.

    b. Snapshots may be aligned to a clock synchronized within their own network [NIST 2015]. A global clock may be too burdensome for sensor networks that operate in the wild. Others, however, argue in favor of a global clock [Li 2004]. We do not endorse either scheme.

[3] *Trustworthiness* includes attributes such as security, privacy, reliability, safety, availability, and performance, to name a few.

c. NoTs may affect business performance – sensing, communicating, and computing can speed-up or slow-down a NoT's workflow and therefore affect the "perceived" performance of the environment it operates in or controls.

d. Snapshots maybe tampered with, making it unclear when events actually occurred, not by changing time (which is not possible), but by changing the snapshot at which an event in the workflow triggers, e.g., sticking in a **delay()** function call.

e. Reliability and performance of a NoT may be highly based on (d).

4 Additional Considerations

Three additional considerations include:

1. **Open, Closed**

   NoTs can be open or closed. For example, an automobile can have hundreds of sensors, numerous CPUs, databases such as maps, wired communication channels throughout the car, and without wireless access between any 'thing' in the car to the outside. This illustrates a closed NoT. Such a NoT mitigates wireless security concerns such as remotely controlling a car, however there could still be concerns of malware and counterfeit 'things' that could result in reduced safety. A fully open system would essentially be any 'thing' interoperating with any 'thing,' anyway, and at any time. This, from a "trustworthiness" standpoint is impossible to assure since the NoT is unbounded.

   Most NoTs will be between these extremes. The primitives serve as a guidepost as to where reliability and security concerns require additional mitigation, e.g., testing.

2. **Patterns**

   We envision a future demand for design patterns that allow larger NoTs to be built from smaller NoTs, similar to design patterns in object-oriented systems. In essence, these smaller entities are sub-NoTs. Sub-NoTs could speed-up IoT adoption for organizations seeking to develop IoT-based systems by having access to sub-NoT catalogues. Further, the topology of sub-NoTs could impact the security and performance of composite NoTs.

3. **Composition and *Trust***

   To understand the inescapable *trust* issues associated with IoT, consider the attributes of the primitives and elements shown in Table 1. The three rightmost columns are our best guess as to whether the pedigree, reliability, or security of an element or primitive creates a trustworthiness risk.

The following table poses questions such as: *what does trust mean for a NoT when its abstractions are in continual flux due to natural phenomenon that are in continuous change and while its virtual and physical entities are unknown, partially unknown, or faulty? Or if we have insecure physical systems employing faulty snapshots composed with incorrect assumed environments, where is the trust?*

Table 1: Primitive and Element Trust Questions

Primitive or Element	Attribute	Pedigree Risk?	Reliability Risk?	Security Risk?
Sensor	Physical	Y	Y	Y
Snapshot	Natural phenomenon	N/A	Y	?
Aggregator	Virtual	Y	Y	Y
Communication channel	Virtual and/or Physical	Y	Y	Y
***e*Utility**	Virtual or Physical	Y	Y	Y
Decision trigger	Virtual	Y	Y	Y
Geographic location	Physical (possibly unknown)	N/A	?	?
Owner	Physical (possibly unknown)	?	N/A	?
Environment	Virtual or Physical (possibly unknown)	N/A	Y	Y
Cost	Partially known	N/A	?	?
Device_ID	Virtual	Y	?	Y

Such questions demonstrate the difficulty of IoT trustworthiness.

An accepted definition of IoT is necessary before we define IoT trustworthiness. Until that definition occurs, the following statement about IoT trustworthiness is:

Trust in some NoT *A*, at some snapshot *X*, is a function of NoT *A*'s assets ϵ {aggregator(s), communication channel(s), *e*Utility(s), decision trigger(s)} with respect to the members ϵ { sensor(s), geographic location, owner, environment, cost, Device_IDs} when applicable.

5 Summary

We presented a common vocabulary to foster a better understanding of IoT. Five primitives and six elements that impact IoT trustworthiness were proposed. The primitives are the building blocks; the elements are the less tangible trust factors impacting a NoT. Primitives also allow for analytics and formal arguments of IoT use case scenarios. Without an actionable and universally-accepted definition for IoT, the model and vocabulary presented here still expresses how IoT *behaves*.

Use case scenarios employing the primitives should afford us quicker recommendations and guidance concerning a NoT's potential trustworthiness. For example, authentication can be used in addressing issues such as geo-location and sensor ownership, but authentication may not be relevant if an adversary owns the sensors and can obtain that information based on proximity. Encryption can protect sensor data transmission integrity and confidentiality including cloud-to-cloud communication, but it might render the IoT sensors unusable due to excessive energy requirements. While fault-tolerant techniques can alleviate reliability concerns associated with inexpensive, replaceable, and defective third party 'things', they can also be insecure and induce communication overhead and increased attack surfaces. In short, primitives and how they can be composed create a design vocabulary for how to apply existing technologies that support IoT trustworthiness. These primitives are simply *objects with attributes*, with the five forming a design *catalog*.

We acknowledge that there may be better labeling for the elements and primitives, and that even a reduction or increase in the number of them, depending on perspective, could prove beneficial. For example, should actuators be primitives in a manner similar to sensors? Or should actuators be treated as a part of the environment element? This model, as it stands, treats actuators as "consumers" of the outputs from decision triggers. Actuators are 'things,' but not all things are individual primitives in this model. This model does however, allow actuators to be classified as *e*Utilities if they feed information back into a NoT's workflow and dataflow.

Future work will involve refining and decomposing the primitives since they are currently abstract and at a high level. The same will occur for several of the elements.

So is IoT simply a handful of applied *systems engineering* principles inside of a distributed system? The answer is not clear, but what is clear is that a composability science is necessary before we can deploy NoTs, with trust. Primitives appear to offer that science and that beginning point.

We hope that the readers will take the opportunity to send feedback to iot@nist.gov on these ideas.

Appendix A—References

[Li 2004] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, Hong Kong, March 7-11, 2004, pp. 564-574. http://dx.doi.org/10.1109/INFCOM.2004.1354528.

[NIST 2015] M. Weiss, J. Eidson, C. Barry, D. Broman, L. Goldin, B. Iannucci, E. A. Lee and K. Stanton, *Time-Aware Applications, Computers, and Communication Systems (TAACCS)*, NIST Technical Note (TN) 1867, National Institute of Standards and Technology, Gaithersburg, Maryland, February 2015, 26pp. http://dx.doi.org/10.6028/NIST.TN.1867.

[Stouffer 2015] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams and A. Hahn, *Guide to Industrial Control Systems (ICS) Security*, NIST Special Publication (SP) 800-82 Revision 2, National Institute of Standards and Technology, Gaithersburg, Maryland, May 2015, 247pp. http://dx.doi.org/10.6028/NIST.SP.800-82r2.